P502492.PDF [Page: 3 of 42]

DEPARTMENT OF NATIONAL DEFENCE
CANADA

OPERATIONAL RESEARCH DIVISION

AORT RESEARCH NOTE RN-9702

A DESCRIPTION OF THE GENERIC SIMULATION USED FOR ESTIMATION OF AIRFIELD CAPACITY AT A FLYING SCHOOL (U)

by

MS A. CHOUINARD
and
DR B.U. NGUYEN

APRIL 1997

OTTAWA, CANADA

National Defence Défense nationale

OPERATIONAL RESEARCH DIVISION

CATEGORIES OF PUBLICATION

ORD Reports are the most authoritative and most carefully considered publications of the DGOR scientific community. They normally embody the results of major research activities or are significant works of lasting value or provide a comprehensive view on major defence research initiatives. ORD Reports are approved personally by DGOR, and are subject to peer review.

ORD Project Reports record the analysis and results of studies conducted for specific sponsors. This Category is the main vehicle to report completed research to the sponsors and may also describe a significant milestone in ongoing work. They are approved by DGOR and are subject to peer review. They are released initially to sponsors and may, with sponsor approval, be released to other agencies having an interest in the material.

Directorate Research Notes are issued by directorates. They are intended to outline, develop or document proposals, ideas, analysis or models which do not warrant more formal publication. They may record development work done in support of sponsored projects which could be applied elsewhere in the future. As such they help serve as the corporate scientific memory of the directorates.

ORD Journal Reprints provide readily available copies of articles published with DGOR approval, by OR researchers in learned journals, open technical publications, proceedings, etc.

ORD Contractor Reports document research done under contract of DGOR agencies by industrial concerns, universities, consultants, other government departments or agencies, etc. The scientific content is the responsibility of the originator but has been reviewed by the scientific authority for the contract and approved for release by DGOR.

DEPARTMENT OF NATIONAL DEFENCE

CANADA

OPERATIONAL RESEARCH DIVISION

AORT RESEARCH NOTE RN-9702

A DESCRIPTION OF THE GENERIC SIMULATION USED FOR ESTIMATION OF AIRFIELD CAPACITY AT A FLYING SCHOOL (U)

by

MS A. CHOUINARD
and
DR B.U. NGUYEN

Recommended by: _____ AORT TEAM LEADER

Approved by: _____ DOR(CAM)

Directorate Research Notes are written to document material which does not warrant or require more formal publication. The contents do not necessarily reflect the views of ORD or the Canadian Department of National Defence.

OTTAWA, CANADA

APRIL 1997

102-02342/

ABSTRACT

This research note documents the simulation used to determine the maximum student capacity for a flying school. The simulation reproduces six different level flying courses at an airport of two runways. There are two main outcomes of the simulation: the average number of required flights that are not completed by the student per course, and the number of double flights (two flights in the same day and by the same student) that have to be done by the students in order to satisfy their requirements.

RÉSUMÉ

Cet article documente une simulation pour déterminer le nombre maximal d'étudiants permis dans une école de vol. Cette simulation reproduit six cours de niveau différent dans un aéroport possédant deux pistes. Il y a deux résultats principaux du programme: le nombre moyen de vols qui ne sont pas complétés par étudiant par cours, et le nombre de vols doubles (deux vols dans la même journée par le même étudiant) que doivent faire des étudiants afin de satisfaire aux exigences du cours.

TABLE OF CONTENTS

ABSTRACT i

RÉSUMÉ i

TABLE OF CONTENTS ii

I-INTRODUCTION 1

II-GENERAL MODEL 1

III-SIMULATION ASSUMPTIONS 4
Level of detail 4

IV-SIMULATION METHODOLOGY 5

V-SOFTWARE REQUIREMENT 6

VI-MODIFICATIONS NEEDED FOR NEW SCENARIOS 7

VII-SUMMARY 7

ANNEX A: USER'S GUIDE A-1

ANNEX B: PASCAL CODE B-1
Simulation Hierarchy B-1
Input file B-5
Global Variables (in alphabetical order) B-5
Subroutines B-8
Unit Upday B-16

A DESCRIPTION OF THE GENERIC SIMULATION USED FOR ESTIMATION OF AIRFIELD CAPACITY AT A FLYING SCHOOL

I-INTRODUCTION

1. This research note describes the simulation used to determine the maximum student capacity at a flying school. The limitation to the number of students that can be enrolled is the number of aircraft which can circle the airfield simultaneously, the number of take-offs and landings that can be performed, the number of aircraft that can do a runway approach in a given time period, and the weather conditions.

II-GENERAL MODEL

2. The outcome of the simulation consists of two statistics. The first is the average number of uncompleted flights per serial. Each student is required to perform a number of flights in his course. For different possible reasons such as air traffic or illness, a student might not be able to complete his requirements. The number of uncompleted flights indicates mainly the level of activity in the air space above the airfield. The second is the number of double flights per student. When a flight is missed by a student, a make-up flight must be performed by flying two missions in a single day, or doubling-up. This statistic measures scheduling problems when the airfield is near saturation. Therefore, it can be used to compare scheduling difficulties between different options.

3. There are six different courses named A to F in the simulation. Each course corresponds to a different level of expertise of the students. For example, there are courses for beginners and there are courses for recertification. For each course, there are several types of missions: clearhood, instrument, navigation, night, and formation. Operations of courses B to F are constrained by the maximum number of students, the maximum number of aircraft, the maximum number of aircraft in a circuit at a time, the maximum number of approaches and take-offs, and the weather condition. In addition to these conditions, course A does check if the prerequisite to a flight for each student is satisfied, e.g., a clearhood mission might have to be performed before a student can fly a navigation mission.

4. There are two runways in the simulation used for the Moose Jaw study. The runways are shown in Figure 1 (not exact in scale). The number of runways is important because it provides a constraint of the air traffic in the sense that the inner runway can handle only 8

aircraft whereas the outer runway can handle 10 aircraft for take-off or landing at a time.

5. Circuit Usage. Each flight segment is encoded into three categories, with respect to circuit usage.

a. Full use of the circuit. The student is required to follow the full path of the circuit without completing a full landing.

b. Approaching the runway. The student approaches the runway from a point outside the circuit. They either land the aircraft, or approach the runway without landing, then immediately leave the circuit.

c. Remote flying. The aircraft does not at any time interfere with the circuit during this phase.

6. There are several profile types. Two flights are of the same profile type if the sequence of flight segments, as defined above, with respect to circuit usage are similar. For example, a typical profile type is a sequence of three phases: take-off, fly to the area and return to the circuit for landing. A flight profile is a sequence of n integers where n is the mission length in units of 15 minutes. Each integer is a code for each different phase: aircraft during take-off, aircraft during approach, aircraft is in the circuit or aircraft is not in the circuit. These integers are used when establishing priorities of a flight. For example, when the aircraft is not in the circuit, its profile is zero indicating a low priority.

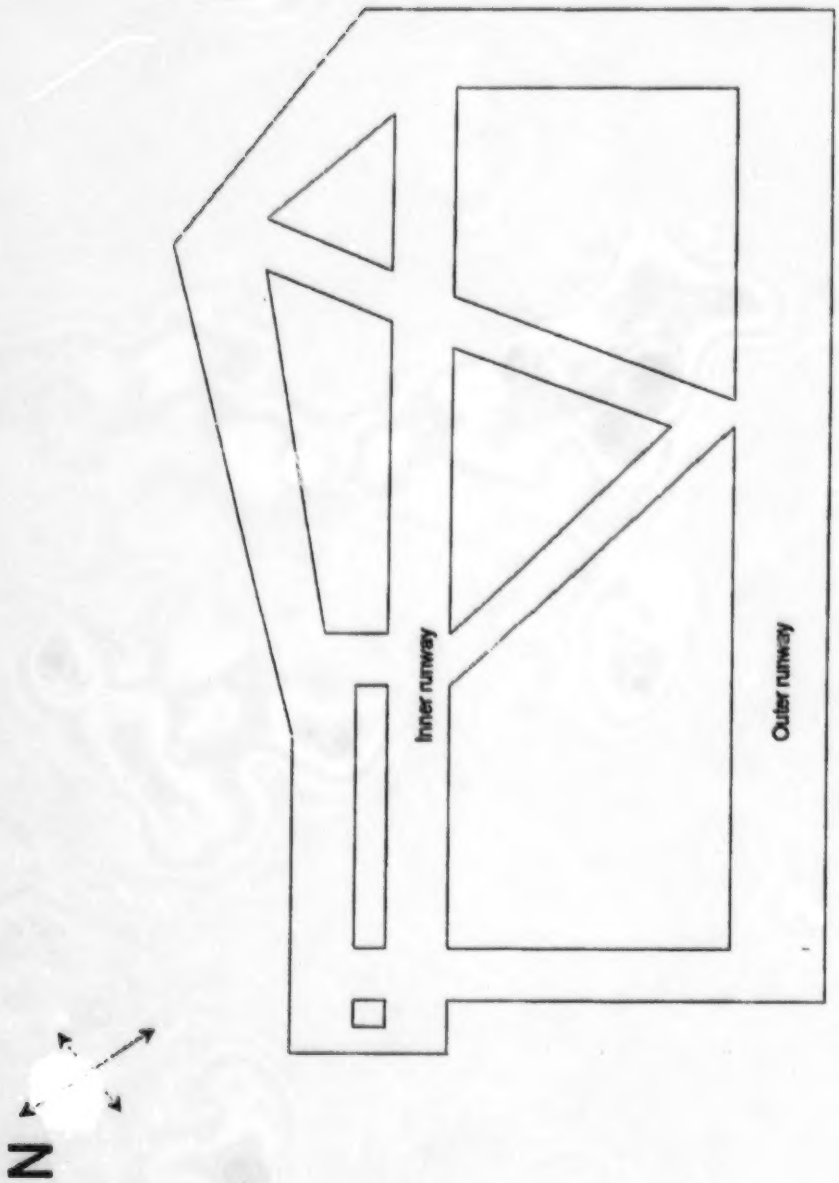

Figure 1 - Runways at Moose Jaw

III-SIMULATION ASSUMPTIONS

7. For the purpose of this study, the following assumptions are made in the design of the simulation (circuit usage, number of approaches and number of take-offs are estimated values):

a. circuit usage is limited to a maximum of 18 aircraft simultaneously, 8 for inner pattern and 10 for outer pattern;

b. 23 approaches to the runway can be made in each 15 minute time period, 8 for the inner pattern and 15 for the outer pattern;

c. a maximum of 15 take-offs, 15 for the inner pattern and 0 for the outer pattern;

d. all courses are operating at maximum capacity;

e. the airfield operates on a nine-hour a day schedule for five days a week;

f. no scheduled flights on evenings and weekends except for specified night flights. However, weekends are occasionally used for catch-up when a student has fallen behind;

g. the number of aircraft is unlimited (but the program can be modified to accommodate a maximum number of aircraft);

h. and the rate of success of the students is not determined.

8. Level of detail:

a. This program tracks every student passing through Moose Jaw in a year, day by day in his training program, e.g., the program tracks that student X enrolled on course B is doing mission instrument 7 on day 112.

b. The program tracks every mission to the 15-minute segment, e.g., when X is

doing mission instrument 7, the program knows whether his plane is in the circuit, taking off or landing.

c. The weather is randomly selected each day to simulate actual conditions, and the simulation is repeated with the same inputs as many times as desired.

IV-SIMULATION METHODOLOGY

9. The simulation steps through each calendar year on a daily basis. At the starting date of a serial, the simulation initialises all the parameters of that serial such as the number of flights to be done for each student in each course and the number of days left in a serial.

10. The analysis proceeds by choosing a random number between 0 and 1. If, for example, if we were analyzing an advanced course which utilizes 8 flying days out of 21 working days (38%) for the month of January, we would expect a good flying day if the random number was less than or equal to 0.38 (and conversely, a bad flying day if it was greater than 0.38), i.e., the probability of having favourable flying weather on any working day in January is 0.38. This probability is obtained from the Air Training Programming Calendars and varies for each month. Similar calculations are produced for the other courses and months. The weather condition for each day is determined and stored for the whole calendar years at the beginning of each run.

11. Once the weather for the day is determined, each student is associated with a flight, for each day if he has not completed his requirement during the time length of his course. The mission for each student is chosen in order from 1 to the maximum number of missions, i.e., if mission 1 is not suitable then the code goes to mission 2 and so on until a suitable mission is found or all the missions have been exhausted. A student might be scheduled with two flights in the same day if he was behind regular schedule for whatever reason. When a flight is put in a stack for a student, depending on his course, the program might check if the prerequisite to the proposed flight is satisfied. The simulation then prioritizes each of the flights for the day. It does that by taking into account the profile of a flight and the ratio of the required number of flights to the remaining time in the course. For each 15-minute period in a two hour block of a flying day, the simulation then schedules as many flights as possible while respecting the air traffic constraint such as the maximum number of aircraft in the circuit at the same time or the maximum number of approaches on each runway.

12. Once all the possible flights are scheduled, the program updates the status of the mission in the sense that it counts the number of flights that are behind regular schedule, it counts also the number of flights done on a weekend for each course, each serial and each student. For example, if a make-up flight was done in course A then the variable DoubleUp_A - number of double flights in course A - is incremented by 1. The program also keeps track which flight is done by which student so that a student does not have to do the same flight twice.

13. The program updates the number of days left in each serial every day of the calendar years. If it is a holiday (non-flying day) then the simulation scrolls to the next flying day and repeat the process starting at paragraph 10 above, or stops if it is the end of the simulation time length.

14. A student might fall behind for several reasons. Bad weather may cancel an entire serial. A student may miss a scheduled flight because of illness. Scheduling conflicts may prevent completion of a flight. When such situations happen, the affected student is forced to make up a flight by doubling-up, that is, they must fly two flights in a single day; the flight generator creates two missions for the student, which the program attempts to schedule.

15. For each course, a student is required to perform a required number of flights. When this requirement is not met, the number of uncompleted flights is collected at the last day of each serial. In reality, the duration of the serial may be extended, but when this number gets large, it is an indication that there are too many students attending the course.

16. The number of flights doubled up per student for each course is also collected. When the airfield is close to its saturation point, the main reason for a student missing a flight is due to scheduling conflicts. The number of flights that have to be doubled up is used as a second indicator for scheduling difficulties which arise when there are too many students.

V-SOFTWARE REQUIREMENT

17. The program is written in Turbo Pascal. There is one Pascal unit, Upday.pas, which needs to be compiled before one compiles the main program Jaw.pas (Annex A). One way to compile Upday.pas is to open Upday.pas by typing "turbo Upday.pas" at the prompt, then click on "Compile" and choose the option "Compile". To execute Upday.pas as a unit after

compiling, the user can click on "Compile" and choose the option "Make" while setting "Destination" to "Disk". The user can now compile Jaw.pas in the same way he does with Upday.pas and the program will be ready to run.

18. Note that the program also expects an input file, Profile.dat, in the working directory. Profile.dat contains information about the flights: profile type, profile, mission length and prerequisite. The details are explained in Annex B.

19. The output data files, NFND_XX.dat, WKENDXX.dat and DBL_XX.dat - the number of flights uncompleted, the number of weekend flights and the number of double flights for course XX, are written to the directory "C:\Jaw". More details are given in Annex A.

VI-MODIFICATIONS NEEDED FOR NEW SCENARIOS

20. To use this program for a new study, the followings are needed:

a. schedule of serials (in subroutines NewCourse and MidCourse);

b. airfield limitation (in unit Upday);

c. calendar of holidays (in subroutine CalcGen);

d. and Profile.dat input file based on phasing charts.

VII-SUMMARY

21. The simulation Jaw.pas described in this research note is a reliable and useful tool for airfield capacity studies. As the program simulates day-by-day events, it describes a realistic operational airfield with influences from weather, holidays and student sick leaves etc. The code has been modified in such a way that it can easily be adapted for possible future studies.

ANNEX A: USER'S GUIDE

1. There are two Pascal program files, Jaw.pas and Upday.pas. Jaw.pas is the main program. Upday.pas contains a variable list and subroutines for the five serial per year. To change the scenario, the following variables might need to be changed.

   a. MaxNumTO: the number of takeoffs which are possible in a 15 minute period.

   b. MaxNumApp: the maximum number of instrument approaches which can be made within a 15 minute period. This is set to two for each runway equipped with instrument approach equipment.

   c. MaxInCct: the maximum number of aircraft allowed in the circuit simultaneously. The limit per runway is five.

   d. Prob_of_missing: the probability that a student will miss a flying day because of illness or other external cause.

   e. MaxNumAC: the maximum number of aircraft available.

   f. WeekendFltNum: the number of flights the students must have fallen behind by before weekend flying is permitted.

   g. NumPeriodsInDay: the number of working 15-minute periods in the day. The default is 36 or nine hours.

2. Once the above variables are set, the main program Jaw.pas will need to be modified.

   a. First_A: the beginning value for the number of students for course A.

   b. Last_A: the largest number of students to be loaded into course A.

c. A_inc: the step value for the simulation from the beginning value, First_A, to the largest number of students, Last_A.

d. Num_Iterations: the number of times the simulation repeats itself for each different set of random numbers.

3. The output is written to a pair of text files called NFND_A.DAT, WKENDA.dat and DBL_A.DAT contained in the directory C:\JAW. The fist file contains the total number of uncompleted flights per year for each iteration of the scenario. The second contains the number of flights during weekend for each iteration. The third contains the number of times a student performs two flights in one day for each iteration.

ANNEX B: PASCAL CODE

Simulation Hierarchy

1. The simulation hierarchy is shown on Figure 2. The two more complicated procedures - Schedule and UpdateDay - are shown on Figures 3, and 4 respectively.

2. First level

   Main program calls InitialMaxApp, Initial, SetDaysLeft, Calgen, UpdateDay, GenerateFlights, AssignPriority, Schedule, SetProfile, UpdateCards.

3. Second level

   -Calgen calls Calcday.

   -Updateday calls MidCourse, NewCourse, UpdateDaysLeft.

   -Generateflights calls Loadflts_DorF, Loadflts_CorE and Loadflts_AorB.

   -Schedule calls Initial, InitialConflict, InitNumApp, Getflight and Sort.

4. Third level

   -Newcourse calls Resetcards, InitA and ResetFltsback.

   -GetFlight calls Updateallowed.

Figure 2 - Simulation Hierarchy

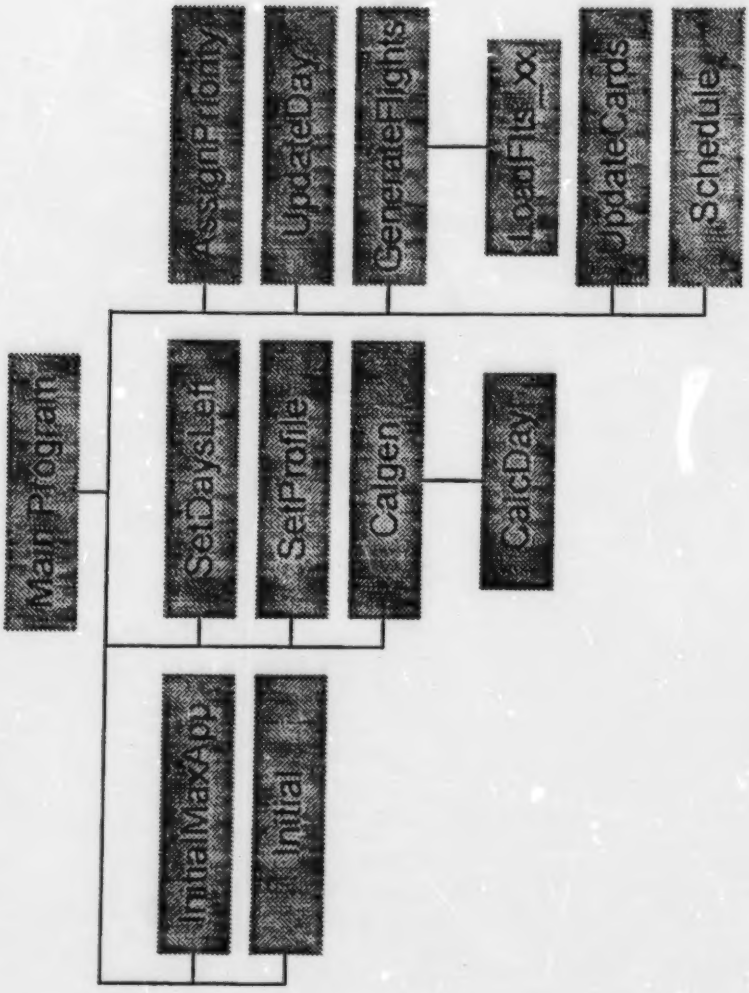

Figure 3 - Schedule

Figure 4 - UpdateDay

Input file

5. An input file of the name Profile.dat is expected by the program in the working directory. The first few lines of Profile.dat give the profile type for each mission. There is one column for each course, and as many rows as there are missions in the longest course. The jth column and ith row is read in as Prof_Type[j,i] and indicates the profile type for the jth course and ith mission. Note an entry of 0 is required where there is no flight,e.g., if course B has 40 missions, but there are 60 rows in the matrix, the column for course B must have an entry of 0 in rows 41 through 60.

6. Immediately after this matrix in Profile.dat is another matrix Profile[j,i]. The ith row in this matrix gives the profile in detail for mission i. For example, if course A mission 7 has Prof_Type[1,7]=13, then Profile[j,13] gives the profile of that mission, broken down into nine 15 minute segments. The value of Profile[j,i] is either 0, 1, or 3 indicating no circuit work, a takeoff or approach, and full circuit work, respectively. The tenth entry in each row, Profile[10,i], gives the actual number of 15-minute segments in the mission.

7. The last array in Profile.dat gives the flight prerequisites, and is read into the array Prereq[j,i]: Prereq[2,i] is a prerequisite to the mission Prereq[1,i].

Global Variables (in alphabetical order)

8. Calendar[j]: jth day of the simulation. It indicates the day code.

9. Day: the day of the simulation calendar in number.

10. Day_code:

1 weekend = weekend
2 weather such that courses C, D and F can fly = fis_recme
3 weather such that courses C, D, E and F can fly = fis_rec
4 weather such that all can fly = aft_plus
5 wind are such that runway M will be used = windy1
6 no flying, bad weather or holiday = noflying
7 wind are such that runway N will be used = windy2

11. Days_Left[i,j]: ith course, jth serial. Remaining days for that serial.

12. Doubleup_XX: number of doubled up missions for course XX.

13. Flight[i,j]: array of student flights to be scheduled.
i=1=student; Flight[1,j] = student id number;
i=2=course; Flight[2,j] = course number;
i=3=serial; Flight[3,j] = serial number;
i=4=mission; Flight[4,j] = mission number;
i=5=priority; Flight[5,j] = priority number;
i=6=used; Flight[6,j] = 1 if the flight is scheduled;
i=7=timeslot; Flight[7,j] = 1, 2, 3 (all the Flight[7,j] = 1 are to be scheduled first then the one with Flight[7,j] = 2. All others are Flight[7,j] = 3.);
i=8=allowed; Flight[8,j] = time at which that flight starts in unit of 15 minutes.

j=flight number

14. Flights_Left[i,j,k]: kth serial, jth course and ith student. Number of remaining flights to be completed.

15. FltsBack_XX[i,k]: number of missions student i is running behind in serial k for course XX.

16. Max_A: maximum number of students in course A.

17. MaxNumApp: maximum number of approaches in a 15 minute period.

18. MaxNumInCct: maximum number of aircraft in circuit at the same time.

19. MaxStudent_Course[i]: maximum number of students allowed in course i.

20. Misslength[i]: ith mission, length in unit of 15 minutes.

21. NFND[i]: number of uncompleted flights for course i for all serials and all students.

22. NumAC[i]: number of aircraft flying at the ith period of the day.

23. Num_Flights: number of flights stacked in the Flight array for the day.

24. NumFlights_XX: the number of missions in course XX.

25. NumFlts_Course[i]: number of mission in course i.

26. NumME: number of multi engine aircraft used at a given time of the day.

27. NumRW: number of rotary wing in use at a given time of the day.

28. NumSerial_Course[i]: number of serials in course i.

29. NumTO[i]: number of take-offs at the ith period of a day.

30. Prereq[i,j]: Prereq[2,j] is the prerequisite for mission Prereq[1,j].

31. Profile[j,i]: jth period of the mission, and ith profile type. Profile is equal to 1 for take-off or approach, 3 if the aircraft is in the circuit and 0 if the aircraft is away from the circuit.

32. Prof_Type[j,i]: jth course, and ith flight. For example, mission number 5 has profile number 20.

33. Rand: random number.

34. StudCard_XX[i,j,k]: this array element is true or false, it is true when mission j has been completed by student i, in serial k, for course XX.

35. Wkend_XX: number of flights on weekends for course XX

Subroutines

36. Procedure UpdateCards

Input: Flight[i,j].

Output: Flights_Left[i,j,k], FltsBack_XX, DoubleUp_XX, Wkend_XX and StudCard_XX for course XX.

Procedures called none.

Called from the main program

Algorithm: UpdateCards goes through all the flights that were done for the current day. When a flight is done for the day UpdateCards updates the following information: the number of flights left to be done, the number of flights behind schedule, the number of double flights, the number of flights done on a weekend for each serial, each course and each student, and which mission has been done by which student. For example, if a flight is a weekend flight for course XX then the variable Wkend_XX is incremented by 1, or if it is a double flight then the variable DoubleUp_XX is incremented by 1.

37. Procedure AssignPriority

Input: Flight[i,j], Prof_Type[i,j], Flights_Left[i,j,k], Days_Left[i,j],Profile[i,j].

Output: Flight[x,y] is returned with Flight[fpriority,j]=priority level.

Procedures called: none

Called from the main program

Algorithm: For all flights stacked in Flight array (for a given student, course, serial) is assigned a priority level as follows:

a. priority level is set to the sum of the digits in the mission profile.
b. if mission profile starts with 1 (take-off or approach), an extra point is allocated.
c. the ratio of the number of flights remaining to be done by that student (for that course and serial) to the number of days left for that course and serial, i.e., if a course is almost finished, the priority is given to that course.

38. Procedure LoadFlts_DorF

Input: StudCard_XX[i,j,k], NumFlts_XX, Day_Code, FltsBack_XX[i,k], Days_Left[i,j].

Output: Flight[x, y] is returned where more flights for courses D or F are loaded.

Procedures called: none.

Called from GenerateFlights.

Algorithm: First, the procedure checks if that student has done mission j and that mission j is less than the number of mission in that course. Once a mission that has not yet been completed by this student is found, if it is the weekend, unless the student is behind by a large number of flights, no flying will be done. If it is a week day, it is determined whether or not the student is absent, if he is then he gets behind by one mission. If all conditions are met, the subroutine inserts a flight for that student in the Flight array. Then for the same student, if the student has mission(s) behind, the program looks for the next mission uncompleted for that student. When it is found, then the time for the first mission for that student is set to 1, and the next mission is inserted into the Flight array, the student is given the opportunity to fly 2 missions that day. Then the same is done for another course. This procedure is repeated for all serials and all students in the course.

39. Procedure LoadFlts_CorE

Input: StudCard_XX[i,j,k], NumFlts_XX, Day_Code, FltsBack_XX[i,k], Days_Left[i,j].

Output: Flight[x, y] is returned where more flights for the courses C or E are loaded if possible.

Procedures called: none.

Called from GenerateFlights.

Algorithm: similar to LoadFlts_DorF

40. Procedure LoadFlts_AorB

Input: StudCard_XX[i,j,k], NumFlts_XX, Day_Code, FltsBack_XX[i,j], Days_Left[i,j], Prereq[i,j].

Output: Flight[x, y] is returned where more flights for courses A or B are loaded.
Procedures called: none.
Called from GenerateFlights.
Algorithm: similar to LoadFlts_DorF except for three things for the course A. First, before a flight is inserted in the Flight array for the course A, it is checked if the prerequisite to that flight has been performed by the student. Second, there is a set of simulator flights which are put in the stack of the Flight array only if there is no other available missions. Third, there is a set of missions for the course A that when assigned, the procedure looks for another - a second mission in the same day.

41. Procedure GenerateFlights

Input: Flight[i,j] and Day_code.
Output: Flight[x,y] where all the possible flights are loaded.
Procedures called: LoadFlts_DorF, LoadFlts_CorE and LoadFlts_AorB.
Called from the main program
Algorithm: simple calls to LoadFlts_XX depending on the day code. For example, if the day code indicates that course A and B can fly that day then GenerateFlights call LoadFlts_AorB to put the students in those courses in the stack of the Flight array whenever possible.

42. Procedure SetDaysLeft

Input: none.
Output: Days_left[i,j].
Procedures called: none.
Called from the main program.
Algorithm: Days_Left is reset to 0 for all courses and all serials at the beginning of each year. (The days left are set to the duration of the course in the procedure UpdateDay in the call statement Newcourse.) A same course may vary in duration for each serial; weekends and holidays are counted in the duration.

43. Procedure Initial

Input: setting.
Output: MaxNumInCct.
Procedures called: none.
Called from the main program.
Algorithm: This procedure sets the array MaxNumInCct to the provided value for the day. MaxNumInCct is the maximum number of aircraft allowed in the circuit at the same time. The number 8 in this procedure represents an extra two hours of operations, that is 8 periods of 15 minutes, that effectively prevent any take-off but allows flights that took off late in the day to be completed.

44. Procedure InitialConflict

Input: setting, setting2.
Output: MaxNumInCct.
Procedures called: none.
Called from the main program.
Algorithm: For the first 8 hours (NumPeriodsInDay) of the day, MaxNumInCct is assigned the value setting. For the 20th period to the 28th period (of 15 minutes), MaxNumInCct is assigned the value setting2. The reason for different maximum numbers of aircraft in circuit for different times of the day is because of weather condition and personnel availability, i.e., lesser visibility at night and less people available after working hours. Note that this procedure is used only for windy days.

45. Procedure InitialMaxApp

Input: none.
Output: MaxNumApp.
Procedures called: none.
Called from the main program.
Algorithm: For all ith 15-minute working periods of the day, MaxNumApp[i] is initialised to the maximum number of approaches allowed at the airport.

46. Procedure SetProfile

Input: none.

Output: Prof_Type[j,i], jth course and ith flight, Profile[j,i], jth period of mission, ith profile type, Misslength[i] ith mission, Prereq[2,i] and Prereq[1,i]

Procedures called: none.

Called from the main program.

Algorithm: These arrays are read from the input file "Profile.dat" installed in the working directory. The matrices XX[j,i] are read by ith row, that is for a given i, data are read for all j. (See also the section input file).

47. Procedure CalGen

Input: none

Output: Calendar[i], ith day of the simulation.

Procedures called: Calcday.

Called from the main program.

Algorithm: At the beginning of each iteration of the simulation, CalGen calls CalcDay for every day of the simulation calendar to determine its day code. If it is a holiday, the day code is equal to zero; otherwise CalGen provides CalcDay with the average number of flying days of each course for a given month so that CalcDay draws out a Monte Carlo day code. A day is denoted by the integer variable n in the program. Thus the 15th day of the calendar years is n = 15.

48. Procedure CalcDay

Input: n, day of the simulation. Numpft, average number of days that all courses can fly that month. Numaft, average number of days that courses C, D, E, and F can fly that month. Numjr, average number of days that courses C, D, and F can fly that month. NumA, average number of days course A can fly that month. Numdays, number of working days for that month.

Output: Calendar[n], nth day of the year.

Procedures called none.

Called from Calgen.

Algorithm: This procedure calculates the weather condition for each day. n=current day. It generates a random number from 0 to 1 to determine the day code. For example, if the random number is greater than or equal to Numpft/Numdays then the day code is such that E can fly. This is because the empirical probability of a flying day in a given month for course E is equal to Numpft/Numdays.

49. Procedure Schedule

Input: Flight[i,j].

Output: Flight[i,j].

Procedures called: Initial, Initialconflict, Initnumapp, Getflight and Sort.

Called from the main program

Algorithm: This procedure schedules the flights for a day. It first initialises NumInCct, NumAC and NumTO using Initial. It initialises MaxNumInCct using InitialConflict depending on the day code. It prioritises the Flight array using Sort. It initialises NumApp using InitNumApp. For all 15-minute periods of the day, it calls GetFlight to schedule all possible flights until saturation is reached, that is when the various constraint parameters, such as the number of aircraft in circuit or the number of take-offs reach their maximum values. This happens when the boolean variable OK is returned as false.

50. Procedure InitNumApp

Input: none.

Output: Numapp[i], ith 15-minute period of the day.

Procedures called: none.

Called from Schedule.

Algorithm: Initialise the number of approaches to zero for the day plus an extra 2 hours in units of 15 minutes. Numapp[i] is the current number of aircraft in circuit at the ith period of the day and because of that it is initialised to zero before any flight is scheduled.

51. Procedure Sort

Input: Flight[i,j].

Output: Flight[i,j] in order of priority

Procedures called none.

Called from Schedule.

Algorithm: This procedure sorts all flights in order of priority. j=1, mission 1 is the highest priority, j=2, mission 2 is the next and so on. If Flight[fpriority, j] is high, that flight has high priority. This procedure makes use of the bubble sort algorithm, that is it goes through a double loop of all missions j and switches two consecutive missions when the order of priority is reversed.

52. Procedure GetFlight

Input: Flight[i,j]. n, nth period of the day. NumInCct[n], number of aircraft in circuit during the nth period. NumRW, number of rotary wing used for the day. NumME, number of multi engine for the day. Ok, a boolean variable.

Output: same as input.

Procedures called: Updateallowed.

Called from: GetFlight determines if a proposed flight satisfies the following constraints: the maximum number of aircraft is not exceeded, the maximum number of aircraft in circuit is not exceeded and the maximum numbers of approaches and take-offs are not exceeded. Note that night flights and simulation missions do not need to satisfy these constraints. These constraints are verified for the next two hours of the time of the day considered. When a flight is allowed for a day, Getflight updates the number of aircraft in the circuit, updates also the number of approaches, increments the number of aircraft of specific type in use (such as multi-engine aircraft), sets flag (used) indicating that flight is scheduled, sets time at which the flight will start.

53. Procedure UpdateAllowed

Input: Flight[i,j], TOD, stnum, Profile[i,j], NumApp[k], MaxNumApp[k], NumInCct[k], MaxNumInCct[k]

Output: Flight[i,j].

Procedures called none.

Called from Getflight.

Algorithm: When a first flight for a student is scheduled (time slot equal to 1), UpdateAllowed is called to stack a second flight for the same student where the time slot is equal to 2. GetFlight will then try to schedule the second flight right at the end of the first flight.

54. Main program

Input: Profile.dat file. Air traffic constraints: maximum number of approaches and maximum number of aircraft in circuit. Weather condition: average number of flying days for each month for each course.

Output: NFND_XX, Wkend_XX, DoubleUp_XX.

Algortihm: Initialises the parameters using the various Initialxx procedures. Determine the day code for each day of the simulation. For each day, it updates the various parameters such as the number of days left in a course using UpdateDay. If it is a non-flying day, UpdateDay goes to the next day until a flying day is found or the program reaches the end of its simulation calendar. It then schedules the most number of flights possible (subject to air traffic constraints and weather condition) using GenerateFlights, AssignPriority and Schedule. At the end of a day UpdateCards updates the status of the missions done. The outputs - the number of flights uncompleted, the number of doubled-ups and the number of flights on weekend - are collected at the end of each run and are written to the output files (xx.dat)

Unit Upday

55. Procedure Init_Const

Input: Max_XX, num_ser_XX, and numflts_XX for course XX.
Output: MaxStudent_Course[i], NumSerial_Course[i], NumFlts_Course[i] and NFND[i].
Procedures called none.
Called from the main program.
Algorithm: The various constants, such as the maximum number of students in a course, are initialised and put in an array to simplify the code. For example, the number of serials in the ith course can be simply accessed by calling NumSerial_Course[i] instead of num_ser_XX where XX is the ith course.

56. Procedure ResetCards

Input: course number; serial, the serial identifier.
Output: StudCard_XX for course XX.
Procedures called none.
Called from NewCourse.
Algorithm: For all students and all missions in the course XX, ResetCards sets the variable StudCard_XX[i,j,k] to false which means that that jth mission has not been done by the ith student of the kth serial.

57. Procedure ResetFltsBacks

Input: course; serial, the serial identifier.
Output: FltsBack_XX; Flights_Left.
Procedures called none.
Called from NewCourse.
Algorithm: Reset FltsBack_XX[i,j] to 0 for all students i, in the given course XX and serial j. Also reset Flights_Left[i,j,k] to the number of missions to be accomplished in this given course j, serial k and all students i in that course.

58. Procedure UpdateDaysLeft

Inputs: none

Output: Days_Left, Day, Flights_Left, NFND_XX for course XX.

Procedures called none.

Called from UpdateDay.

Algorithm: For each course and serial, if it is the last day of a course and the day identifier is greater than 215, that is if seven months have gone by, then count the number of uncompleted flights for each of the courses; also decreases the number of days left by one for the currently running courses.

59. Procedure InitA

Input: serial identifier.

Output: StudCard_A, Flights_Left.

Procedures called none.

Called from Newcourse.

Algorithm: The first 3, 4 or 5 students of a given serial in the course A are credited with 12 missions, that is StudCard_A for these students and these missions are set to true. Consequently, the number of flights left to be done, Flights_Left[i,1,k] (ith student, course A and the given serial k) is lowered by 12.

60. Procedure NewCourse

Input: course number, serial, D, DL.

Output: Days_Left, StudCard_XX, for course XX, FltsBack_XX, for course XX, Flights_Left.

Procedures called: ResetCards, ResetFltsBack and InitA.

Called from UpdateDay

Algorithm: Initialise a new course in the sense that it initialises StudCard_XX to indicate that these are new missions using ResetCards, initialises FltsBack_XX and Flights_Left using ResetFltsBack, and if the new course is the course A then grant the first 3, 4 or 5 students 12 missions using InitA.

61. Procedure MidCourseXX

Input: serial identifier; NumDone.

Output: FltsBack_XX.

Procedures called none.

Called from UpdateDay.

Algorithm: Update FltsBack_XX for all missions of the specified serial that have been done (StudCard_XX[student, mission, serial] is true) by mid course time. The variable FltsBack_XX is set equal to the number of flights to be done in the course less the number of flights that have been done by mid course time. This is repeated for all missions and all students in the course XX. Note that for course A, there is a set of missions (LoadNext_A) that are not counted as flights.

62. Procedure UpdateDay

Input: none.

Output: outputs of all the other procedures that it called.

Procedures called: UpdateDaysLeft, NewCourse, MidCourseXX.

Called from the main program.

Algorithm: For each day of the simulation call UpdateDaysLeft to update the number of days left in the course. If it is the last day of the course then UpdateDaysLeft collects the number of uncompleted flights for each course. If the day code indicates a holiday, then UpdateDay goes to the next day until the day code is a flying day or the calendar years have expired. For each day of the simulation if it is the starting date of a new course, call NewCourse to initialise the parameters, and if it is at mid course time, call MidCourseXX to update FltsBack_XX.

UNCLASSIFIED
SECURITY CLASSIFICATION OF FORM
(highest classification of Title, Abstract, Keywords)

DOCUMENT CONTROL DATA
(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (the name and address of the organization preparing the document. Organizations for whom the document was prepared e.g. Establishment Sponsoring a contractor's report, or tasking agency, are entered in Section 8).

Operational Research Division
Department of National Defence
Ottawa, Ontario K1A 0K2

2. SECURITY CLASSIFICATION (overall security classification of the document, including special warning terms if applicable)

UNCLASSIFIED

3. TITLE (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C or U) in parentheses after the title)

A DESCRIPTION OF THE GENERIC SIMULATION USED FOR ESTIMATION OF AIRFIELD CAPACITY AT A FLYING SCHOOL (U)

4. AUTHORS (last name, first name, middle initial)

CHOUINARD, A. AND NGUYEN, B.U.

5. DATE OF PUBLICATION (month Year of Publication of document)

APRIL 1997

6a. NO OF PAGES (total containing information. Include Annexes, Appendices, etc.)

34

6b. NO OF REFS (total cited in document)

7. DESCRIPTIVE NOTES (the category of document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

AORT RESEARCH NOTE

8. SPONSORING ACTIVITY (the name of the department project office or laboratory sponsoring the research and development. Include the address).

9a. PROJECT OR GRANT NO. (if appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.)

AORT RESEARCH NOTE RN-9702

9b. CONTRACT NO. (if appropriate, the applicable number under which the document was written.)

10a. ORIGINATOR's document number (the official document number by which the document is identified by the originating activity. This number must be unique to this document.)

10b. OTHER DOCUMENT NOS. (Any other numbers which may be assigned this document either by the originator or by the sponsor.)

11. DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification.)

(X) Unlimited distribution
( ) Distribution limited to defence departments and defence contractors: further distribution only as approved
( ) Distribution limited to defence departments and Canadian defence contractors; further distribution only as approved
( ) Distribution limited to government departments and agencies: further distribution only as approved
( ) Distribution limited to defence departments; further distribution only as approved
( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in 11) is possible, a wider announcement audience may be selected.)

UNCLASSIFIED
SECURITY CLASSIFICATION OF FORM

13. ABSTRACT (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual).

This Research Note documents the simulation used to determine the maximum student capacity for a flying school. The simulation reproduces six different level flying courses at an airport of two runways. There are two main outcomes of the simulation; the average number of required flights that are not completed by the student per course, and the number of double flights (two flights in the same day and by the same student) that have to be done by the students in order to satisfy their requirements.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus, e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified . If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

AIRFIELD

RUNWAY

AIRCRAFT

SIMULATION

FLYING SCHOOL